Codec Documentation

Bruce Cain

August 2018

1 Targeted Features

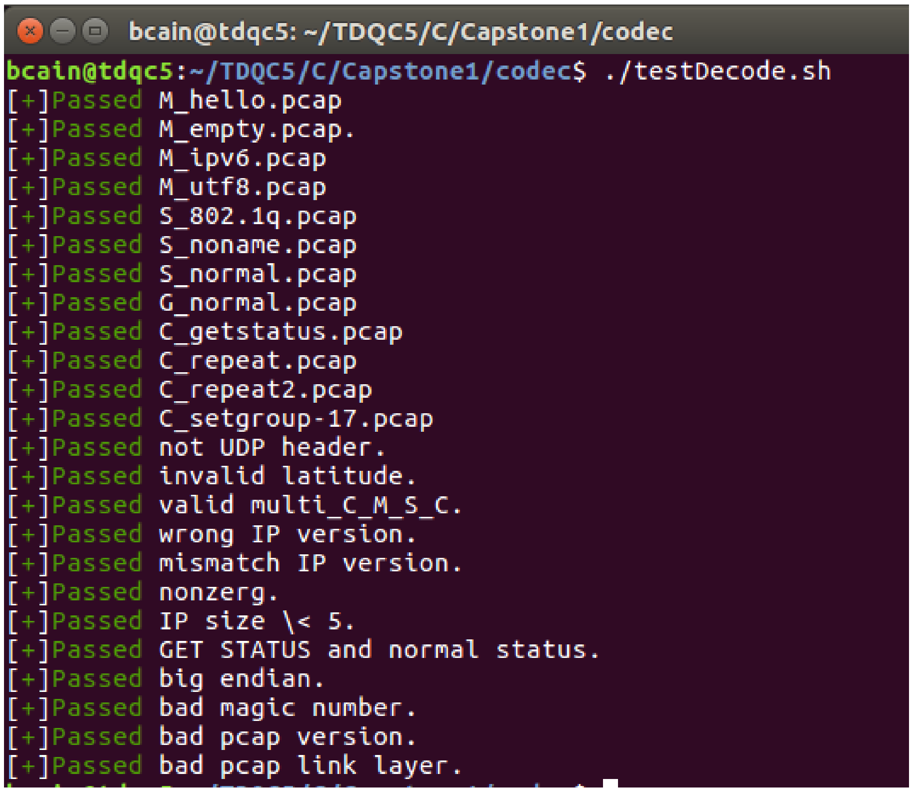
The handling of UTF8 feature was handle the same way as a handling regular ASCII. This feature happened to work along side the basic requirements which made its implementation nice to do. Another feature was accounting for variable size headers such as the Ethernet having a 802.1q header or IPv4 having options. Looking out for IPv6 was another feature implemented which worked well along implementing the variable sized headers. Since several of the fields had to be checked also had to be checked for IPv6. The big endian feature was also implemented, though seemed to effect several other things it did not do as much damage as initially thought.

2 Architecture

All fields found in the pcap file and file to be encoded except the name and message were all stored into structs. This allowed for ease of manageability of information between reading and writing to the standard output or file. both decode and encode follow a similar layout, verify all information is correct then began checking types to see which payload to build or print. This same pattern continues until an error occurs or the end of the pcap/file is found and they exit gracefully.

3 Testing Procedure

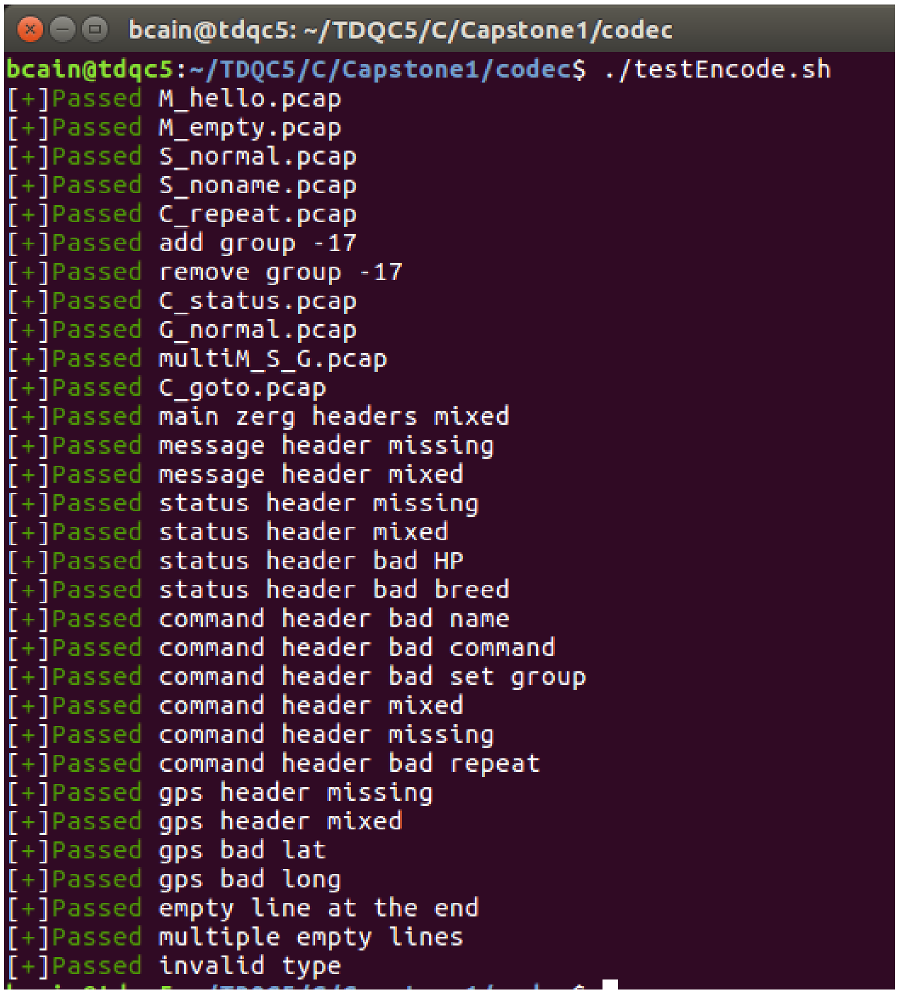
The testing procedures done use two simple bash scripts, one for decode and encode. For decode the script simple reads pcaps from a local directory trying to encode normal and malformed pcaps. The exit codes are compared to expected codes and if correct a [+]Passed ... message will appear. If the code is not expected then [-]Failed ... message appears. Valgrind is also ran to check for memory leaks, and [-]Failed ... message is displayed and the valgrind output is save to a file if the grep fails.



```
bcain@tdqc5: ~/TDQC5/C/Capstone1/codec
bcain@tdqc5:~/TDQC5/C/Capstone1/codec$ ./testDecode.sh
[+]Passed M_hello.pcap
[+]Passed M_empty.pcap.
[+]Passed M_ipv6.pcap
[+]Passed M_utf8.pcap
[+]Passed S_802.1q.pcap
[+]Passed S_noname.pcap
[+]Passed S_normal.pcap
[+]Passed G_normal.pcap
[+]Passed C_getstatus.pcap
[+]Passed C_repeat.pcap
[+]Passed C_repeat2.pcap
[+]Passed C_setgroup-17.pcap
[+]Passed not UDP header.
[+]Passed invalid latitude.
[+]Passed valid multi_C_M_S_C.
[+]Passed wrong IP version.
[+]Passed mismatch IP version.
[+]Passed nonzerg.
[+]Passed IP size \< 5.
[+]Passed GET STATUS and normal status.
[+]Passed big endian.
[+]Passed bad magic number.
[+]Passed bad pcap version.
[+]Passed bad pcap link layer.
```

Figure 1: Decode testing output

The encode testing procedure is the same except the directory has decoded output in a file, there is also another test done with file that are suppose to encode correctly and not error out. A decode command is ran on the built pcap then diff is ran against the original decode to the encode decoded output. If that fails again [-]Failed ... message is printed and the diff output is saved into a file.



```
bcain@tdqc5: ~/TDQC5/C/Capstone1/codec
bcain@tdqc5:~/TDQC5/C/Capstone1/codec$ ./testEncode.sh
[+]Passed M_hello.pcap
[+]Passed M_empty.pcap
[+]Passed S_normal.pcap
[+]Passed S_noname.pcap
[+]Passed C_repeat.pcap
[+]Passed add group -17
[+]Passed remove group -17
[+]Passed C_status.pcap
[+]Passed G_normal.pcap
[+]Passed multiM_S_G.pcap
[+]Passed C_goto.pcap
[+]Passed main zerg headers mixed
[+]Passed message header missing
[+]Passed message header mixed
[+]Passed status header missing
[+]Passed status header mixed
[+]Passed status header bad HP
[+]Passed status header bad breed
[+]Passed command header bad name
[+]Passed command header bad command
[+]Passed command header bad set group
[+]Passed command header mixed
[+]Passed command header missing
[+]Passed command header bad repeat
[+]Passed gps header missing
[+]Passed gps header mixed
[+]Passed gps bad lat
[+]Passed gps bad long
[+]Passed empty line at the end
[+]Passed multiple empty lines
[+]Passed invalid type
```

Figure 2: Encode testing output

4 Write Up

4.1 Goals

Several goals were made through out this project, Completing the base requirements of simply decoding and encoding. Creating the ability to handle different types of packets along with multiple packets. Managing to do some of the extra features and still maintaining the base requirements.

4.2 Challenges/Success

Several challenges throughout this project, reading in a pcap was an entire problem in itself. Not familiar with pcap headers made it difficult to parse, though a quick read up fixed that. Reading in the bytes was not as difficult as originally thought but learning how the host flipped bits around made handling them interesting. Keeping that in mind managing them was quite bearable. Using a union to take in a single or double precision int and out as a float or double was a excellent discovery in itself.

Keeping up with all the freeing needed to be done with as many possible exits throughout the code was somewhat difficult and also made the code messy. Everything is freed but messy code is something to improve on, time was not as long as it felt which made it hard to do things cleanly.

4.3 Lessons Learned

A better design and better testing procedure earlier on would of saved lots of time in the earlier stages of the project. If time allows build those two things out would make keeping up with parts of the code a lot more manageable. Also will build a clear picture of what needs to be done and cleaner code as well. Again the union to manage floats and doubles is something well noted.

4.4 Conclusion

Though this project seemed large and overwhelming in the beginning, it turned out to be a nice two week assignment. Maybe another week to work on improving or possibly implementing better procedures but there was enough time to get everything done and then some. Learned a lot through out the project and not only about programming but about how packets are laid out. Continued work on this project is planned, along with possibly better implementation and practice of better design.